Midterm 1 - Section 001 (C)

Name:

Penn State access ID (xyz1234) in the following box:

Student ID number (9XXXXXXXX):

Instructions:

- Answer all questions. Read them carefully first. Be precise and concise. Handwriting needs to be neat. Box numerical final answers.
- Please clearly write your name and your PSU access ID (i.e., xyz1234) in the box on top of **every page**.
- Write in only the space provided. You may use the back of the pages only as scratch paper. **Do not write your solutions in the back of pages!**
- **Do not write outside of the black bounding box**: the scanner will not be able to read anything outside of this box.
- We are providing two extra pages at the end if you need extra space. Make sure you mention in the space provided that your answer continues there.

Good luck!

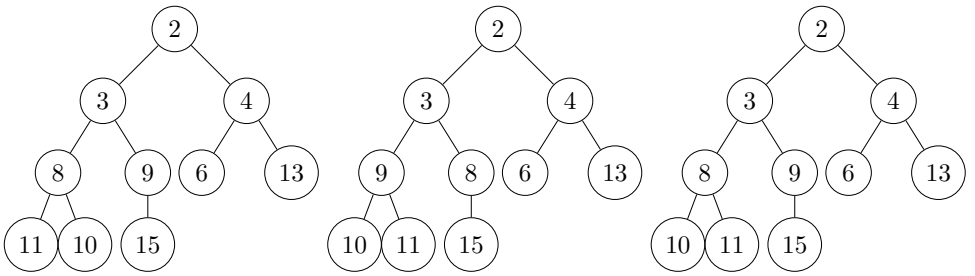
Name:	**PSU Access ID (xyz1234):**

Name:	PSU Access ID (xyz1234):

Multiple choice questions *(27 points)*

For each of the following questions, select the right answer by filling in the corresponding grading bubble grading bubbles.

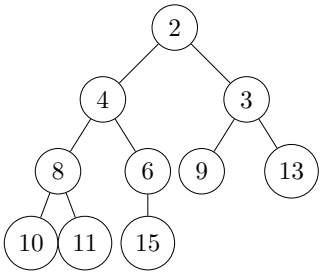
1. Run build-heap on the array $[4, 8, 6, 10, 9, 3, 13, 2, 11, 15]$ to construct a min heap. What would be the resulting min heap?



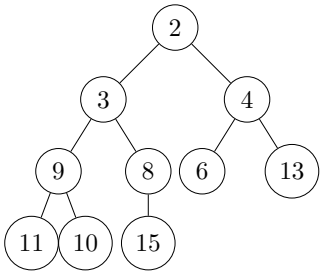
(a)

(b)

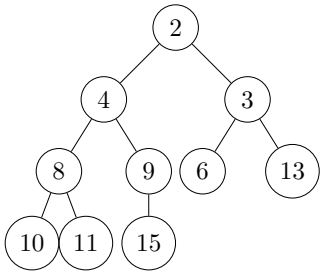
(c)



(d)



(e)



(f)

Answer.

◯ (a)
◯ (b)
◯ (c)
◯ (d)
◯ (e)
● (f)

Name:	**PSU Access ID:**

2. Consider a divide and conquer algorithm that solves a problem of size n by recursively solving two subproblems of size $n-1$ and then combining the solutions in constant $\Theta(1)$ time. What is the running time $T(n)$ of the algorithm?

   ◯ $\Theta(n^2)$
   ◯ $\Theta(\log n)$
   ◯ $\Theta(n)$
   ● $\Theta(2^n)$

3. Given two min heaps of size n each, what is the minimum possible time complexity to make a ONE min-heap of size $2n$ from elements of two given min heaps?

   ◯ $O(n^2)$
   ◯ $O(n \log \log n)$
   ● $O(n)$
   ◯ $O(n \log n)$

4. Consider the recurrence $T(n) = 6T(\frac{n}{3}) + \Theta(\sqrt{n})$. The total work done at a SINGLE sub-problem at level (depth) k of the recursion tree is:

   ◯ $O\left(\frac{n}{3^k}\right)$
   ◯ $O\left(\frac{\sqrt{n}}{3^k}\right)$
   ◯ $O\left(\sqrt{\frac{n}{6^k}}\right)$
   ● $O\left(\sqrt{\frac{n}{3^k}}\right)$

5. For which of the following is $f(n) = \Theta(g(n))$, which of the following is true?

   ● $f(n) = n,\ g(n) = 2^{\log n}$
   ◯ $f(n) = 3^n,\ g(n) = n3^n$
   ◯ $f(n) = n^5,\ g(n) = 8^{\log n}$
   ◯ $f(n) = n \log n,\ g(n) = 2^{\log(5n)}$
   ◯ None of the above

6. For a recursive algorithm with running time $T(n) = 5T\left(\frac{n}{25}\right) + O(\sqrt{n})$, which of the following is true using Master theorem?

   ◯ $T(n) = O(n^2 \log n)$
   ● $T(n) = O(n^2)$
   ◯ $T(n) = O(\sqrt{n} \log n)$
   ◯ $T(n) = O(\sqrt{n})$
   ◯ None of the above

Name:	**PSU Access ID:**

7. We are given an array of n elements which is already a min heap, and would like to convert it to a max heap. What would be the *minimum time complexity*?

   ● $O(n)$
   ◯ $O(n \log n)$
   ◯ $O(n \log n + n)$
   ◯ $O(\log n)$

8. The array [1, 4, 2, 5, 6, 7, 8] corresponds to a min heap. In the space provided below, write down the resulting heap (as an array) after the first two iterations of the heapsort algorithm.

   ◯ [5, 6, 7, 8, 4, 2, 1]
   ● [4, 5, 7, 8, 6, 2, 1]
   ◯ [8, 4, 7, 5, 6, 2, 1]
   ◯ [1, 4, 2, 5, 6, 7, 8]
   ◯ [7, 4, 8, 5, 6, 2, 1]
   ◯ [1, 2, 4, 5, 6, 7, 8]

9. How many integer multiplications does Strassen's algorithm need to do to multiply two 4×4 matrices of integers?

   ◯ 7
   ◯ 16
   ● 49
   ◯ 64
   ◯ 343
   ◯ 407
   ◯ 427

Name:	PSU Access ID (xyz1234):

True/False *(20 points)*

True or false? Fill in the correct bubble. No justification is needed.

T **F**

◯ ● Checking whether a given number (value) exists in a min heap with n elements takes $O(\log n)$ time.
Explanation: There is no order between the left and right children of each node.

● ◯ The procedure heapify-down could make $O(1)$ swaps only.
Explanation: This asks about a possible case and not a general case.

◯ ● In a min heap, the left child of the root is smaller than the right child of the root.
Explanation: The min-heap property does not specify a relationship between the left and right children.

◯ ● $T(n) = 2T\left(\frac{n}{2}\right) + g(n)$ is $\Theta(n)$ for any $g(n) = O(\sqrt{n})$.

● ◯ $2^{c \log_2^n} = \Theta(n^c)$.

◯ ● $2^n = \Omega(n!)$.

● ◯ $3^{\log \sqrt{n}} = \Theta(\sqrt{n})$.

● ◯ $\sum_{i=1}^{\sqrt{n}} i = \Omega(n)$

◯ ● If $f(n)$ and $g(n)$ are non-negative functions, then either $f(n) = O(g(n))$ or $g(n) = O(f(n))$.
Explanation: Consider $f(n) = 1$ when n is even and n^2 when n is odd. Let $g(n) = n$. Then, we can see that $f(n) \notin O(g(n))$ and $g(n) \notin O(f(n))$.

◯ ● If $f(n) = O(g(n))$, then there is some value of n where $f(n) \geq g(n)$.

● ◯ $\sum_{i=1}^{n} 4^i = O(4^n)$.
Explanation: Refer HW1

Name:	PSU Access ID:

Recurrences *(12 points)*

Each of the following scenarios outlines a divide-and-conquer algorithm. In each case, write down the appropriate recurrence relation for the running time as a function of the input size n and give its solution. Giving your solution in O-notation suffices. You need not give a full derivation of your solution if you are using the Master Theorem, but if you need to directly unroll the recursion, you need to show the key steps. You may assume that n is of some special form (e.g., a power of two or some other number) and that the recurrence has a convenient base case with cost $\Theta(1)$.

(a) An input of size n is broken down into 2 sub-problems, each of size $n/2$. The time taken to construct the sub-problems, and to combine their solutions, is $\Theta(n \log n)$.

Solution1: The recurrence relation is $T(n) = 2T(\frac{n}{2}) + \Theta(n \log n)$. Using unrolling, we can get

$$T(n) = 2^{(log_2 n)} T(1) + \Theta(\sum_{i=0}^{log_2 n} n log \frac{n}{2^i})$$

So

$$\begin{aligned} T(n) &= n + \Theta(\sum_{i=0}^{log_2 n} n \log \frac{n}{2^i}) \\ &< n + O\left(\sum_{i=0}^{\log_2 n} n \log n\right) \\ &= n + O\left(n(\log n)^2\right) \\ &= O\left(n(\log n)^2\right) \end{aligned} \tag{1}$$

Solution2: Using the recurrence tree, we can get the running time for each layer as $O(n \log n)$. The number of layers is $\log_2 n = O(\log n)$. So the total running time is $O(n(\log n)^2)$.

(b) An input of size n is broken down into $\sqrt{n}$ sub-problems, each of size $\sqrt{n}$. The time taken to construct the sub-problems, and to combine their solutions, is $\Theta(n)$.

Solution: $T(n) = \sqrt{n}T(\sqrt{n}) + \Theta(n)$. This was essentially Problem 4j from Homework 4. By unfolding that

$$T(n) = n^{\frac{1}{2}+\frac{1}{4}+\cdots+\frac{1}{2^k}} + kn = T(n^{\frac{1}{2^k}}) + kn$$

Now if we let $k = \log_2 \log_2 n$, we have $n^{\frac{1}{2^k}} = 2$ which is constant. Since $\frac{1}{2} \leq \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^k} \leq 1$, we have $T(n) = \Theta(n \log \log n)$.

Name:	PSU Access ID:

Algorithms. *(20 points)*

(a) Given an array A of integers with size n, the the goal is to find k largest elements in the array. A naive solution is to sort the array first and then pick the k largest elements which takes an $O(n \log n)$ time. Using heaps, propose an algorithm with time complexity $O(n \log k)$ to find the k largest numbers in A. Please describe the key steps of your idea at a high level and show its time complexity. You can assume you have access to the heap as a black box.

Solution: This question is similar to problem 5 part 2 on worksheet 3, except that it is asking to find the k largest numbers. Therefore, the solution will use a min heap instead of a max heap, and for each remaining element in the array, we check if it is larger than the value of the root node, if so, we delete the root of the min heap and insert this new element. What's more, the solution returns the entire heap rather than just the root. To see the running time, note that the Build-Heap of $k \leq n$ elements will take $O(k) = O(n)$. Since the heap has k elements, insert and delete take $O(\log k)$, and these operations will be called $n - k = O(n)$ times, giving $O(n + n \log k) = O(n \log k)$ running time.

Name:	**PSU Access ID (xyz1234):**

(b) Suppose you are given an array A with n numbers. You are told that the sequence of values in A is uni-modal: For some index p between 0 and $n-1$, the values in the array entries decrease up to position p in A and then increase the remainder of the way until position $n-1$, thus creating a valley.

You'd like to find the "valley entry" p without having to read the entire array, by accessing as few entries of A as possible. Show how to find the entry p by accessing (reading) at most $O(\log n)$ entries of A.

Solution: To find the valley, set the initial search range from start (left = 0) to end (right = n-1). Keep checking the middle number (mid). If this middle number is less than its neighbors, it's the valley, so return it. If it's not, decide which half to keep searching: if the middle number is in the increasing part, the valley is to the left, so move right to mid+1. If it's in the **decreasing** part, the valley is to the right, so move left to mid-1. With each step, the search range gets smaller by half, which means the time to find the valley is proportional to the logarithm of the array size (O(log n)).

Name:	**PSU Access ID (xyz1234):**

Name:	**PSU Access ID (xyz1234):**

Name:	PSU Access ID (xyz1234):